6. Friendlier

Program Name: Friendlier.java Input File: friendlier.dat

After the success of your "Friendly" encryption that encrypts messages between individuals in your group of friends you have decided to tweak the algorithm to be a bit stronger for the best friends in the group. You used all of your creative juices to come up with the name for the new algorithm: Friendlier.

The messages you and your best friends will be sending are of the form `<NAME>: <MSG>` where `<NAME>` is the first name of the person sending the message and `<MSG>` is a series of ASCII characters which comprises the message.

The first thing the algorithm needs to do is create a seed that will be used to encrypt the text. This is done by starting with the prime number 97 then XOR'ing that value with each letter of the sender's name, from left to right. To encrypt the message, you start by treating the message as a single, large binary value consisting of all the characters in order. Then you take the large binary value and rotate all the bits to the right by 5, making sure that the bits that fall off the right shift back on to the left. Once the message has been rotated you should take each byte and XOR it with the seed, which generates the final encrypted message.

Take, for example the first input value `Bob: :)`

Generate Seed

```
97=0x61=01100001
 B=0x42=01000010
    XOR=00100011
 o=0x6F=01101111
    XOR=01001100
 b=0x62=01100010
seed: XOR=00101110
```

Encrypt message using seed

```
:) = 0x3A 0x29 =  00111010 00101001
Rotate 5 bits from right onto left  01001001 11010001
XOR each byte to seed               00101110 00101110
Encrypted message                   01100111 11111111
                                       0x67     0xFF
```

Input

The input file will contain an unknown number of lines, with one message to encrypt per line.

Output

You should print out one encrypted message per line, including the sender's name followed by a colon and a space. The encrypted text should be printed out as hexadecimal numbers representing the byte value of each ASCII character, with a space after each, as shown below.

Example Input File

```
Bob: :)
Brick: I love lamp!
Ron: I'm Ron Burgundy?
```

Example Output to Screen

```
Bob: 0x67 0xFF
Brick: 0x3A 0x79 0x33 0x53 0x4B 0x83 0x19 0x33 0x53 0x3B 0x5B 0xB1
Ron: 0xC8 0x7B 0x09 0x5B 0x30 0xA1 0x49 0x43 0x30 0x21 0x99 0xA1 0x09 0x99
0x41 0x11 0xFB
```

Notes:

- The encrypted message for `Ron` extends across two lines in the printed version above but constitutes only one line in the actual output.
- A space at the end of each line is optional.